Damien Chiem Exercise 4

MAT 115

Exercise #4

In section 2.9 we start doing some simple examples of data wrangling, i.e., putting data in the form we want in order to eventually summarise, visualize, model, etc.

We start by loading the dslabs package first.

```
library(dslabs)
```

And we load the `exams` dataframe if necessary.

```
load("exams.rda")
```

The book discusses the commands `sort()`, `order()`, and `rank()`. Make sure you know the difference between them and what they do.

Example: we want to know the improvements in exams scores, ordered from lowest to highest. This is what `sort()` does.

```
sort(exams$improve)
```

```
##  [1] -16 -11  -7  -5  -4  -2  -2  -1  -1  -1   0   0   0   0   0   1   2   2   3
```

We see that the largest improvement is only 3 points, but the worst "improvement" is -16 points (out of 50).

We can also sort these numbers from largest to smallest:

```
sort(exams$improve,decreasing=TRUE)
```

```
##  [1]   3   2   2   1   0   0   0   0   0  -1  -1  -1  -2  -2  -4  -5  -7 -11 -16
```

If we want to know *which* students have these improvements, we use `order()`. You might want to read section 2.9.2 in the book. It explains what `order()` does.

```
ind <- order(exams$improve)
exams$ID[ind]
```

```
##  [1] "15" "16" "19" "7"  "9"  "11" "14" "2"  "10" "18" "1"  "5"  "6"  "12" "13"
## [16] "4"  "8"  "17" "3"
```

```
# Order returns the indices of the sort function uses to sort.
```

Student #15 was the one with the disastrous decrease in exam scores, and student #3 improved the most.

We can also see the ranks of the students, say in exam2.

```
rank(-exams$exam2)
```

```
##  [1]  4.0  8.0 15.0  5.5 11.5  1.5 11.5 14.0  8.0  8.0 16.5  1.5  5.5 11.5 18.5
## [16] 18.5  3.0 11.5 16.5
```

```
#exams$exam2
```

We see that student #1 ranks fourth in exam 2 (he/she/they got the fourth highest score in exam 2), while student #6 is tied with student #12 for 1st-2nd (they got the highest scores on exam 2).

Why is there a minus sign? What happens if you omit the minus sign?

Without the minus sign, it would rank based on smallest to largest. With the minus sign it ranks it from largest to smallest.

Use your new skills to find out which student had the highest score on exam 1 and what is the value of that score.

```
# Write code here.
order(exams$exam1)
```

```
##  [1]  3  8 11 16  5  4 18 19  2 10 13 14 15 17  1  6  7  9 12
```

```
sort(exams$exam1)
```

```
##  [1] 39 41 41 42 45 46 46 46 47 47 47 47 47 47 48 50 50 50 50
```

```
print(exams$exam1[12]) # --> highest score
```

```
## [1] 50
```

```
# print(max(exams$exam1)) this also works
```

Sometimes it is useful to have the whole dataframe sorted according to some column. We can do this by using some matrix indexing commands:

```
sortedexams <- exams[order(exams$improve), ]
exams
```

```
##    ID exam1 exam2 improve
## 1   1    48    48       0
## 2   2    47    46      -1
## 3   3    39    42       3
```

```
## 4   4    46    47       1
## 5   5    45    45       0
## 6   6    50    50       0
## 7   7    50    45      -5
## 8   8    41    43       2
## 9   9    50    46      -4
## 10 10    47    46      -1
## 11 11    41    39      -2
## 12 12    50    50       0
## 13 13    47    47       0
## 14 14    47    45      -2
## 15 15    47    31     -16
## 16 16    42    31     -11
## 17 17    47    49       2
## 18 18    46    45      -1
## 19 19    46    39      -7
```

Take a look at the new dataframe called **`sortedexams`** *and compare it with* **`exams`**. *I hope you see what the commands above did. Explain in your own words how the two dataframes differ.*

The improve column has been sorted from least to greatest and exam1 and exam2 columns have been sorted in a way to show that difference between each value in exam1 and exam2 gets you the corresponding value in the improve column. e.g for ID 15: exam1 = 47 and exam2 = 31 31 - 47 = -16.

```
# Write the command to look at `sortedexams` here.
sortedexams
```

```
##    ID exam1 exam2 improve
## 15 15    47    31     -16
## 16 16    42    31     -11
## 19 19    46    39      -7
## 7   7    50    45      -5
## 9   9    50    46      -4
## 11 11    41    39      -2
## 14 14    47    45      -2
## 2   2    47    46      -1
## 10 10    47    46      -1
## 18 18    46    45      -1
## 1   1    48    48       0
## 5   5    45    45       0
## 6   6    50    50       0
## 12 12    50    50       0
## 13 13    47    47       0
## 4   4    46    47       1
## 8   8    41    43       2
## 17 17    47    49       2
## 3   3    39    42       3
```

You should also be aware of the `which.max` and `which.min` commands. To illustrate these commands, we'll use the `stars` dataset. Suppose we want to know the hottest star temperature. That's easy.

```
#First, we fix the wrong classification of `star` and `type`.
stars$star <- as.character(stars$star)
stars$type <- as.factor(stars$type)
```

```
# Then we find the maximum temperature
max(stars$temp)
```

```
## [1] 33600
```

But which star is it? The `which.max` function can tell us. (We can also sort the whole dataset, but if all we want is the maximum, `which.max` works.)

```
stars$star[which.max(stars$temp)]
```

```
## [1] "Alnitak"
```

```
#which.max tells us the name of the max star
```

Find the brightest star. What is its magnitude? (Remember that the brightest star is the one with the most negative magnitude.)

```
#Write your code here.
max(stars$magnitude)
```

```
## [1] 17
```

```
stars$star[which.max(stars$magnitude)]
```

```
## [1] "G51-I5"
```

The section ends with a warning about recycled numbers. R does this when you ask it to do vector operations that should not be done. For example, adding two vectors of different lengths. The situation is actually somewhat worse than the book describes, because R will happily add a 3-vector with a 6-vector without any warning whatsoever.

```
x<-c(1,2,3)
y<-c(10,20,30,40,50,60)
x+y
```

```
## [1] 11 22 33 41 52 63
```

```
# extends shorter vector by repeating original values until same length as longer vector
```

Section 2.13 introduces techniques that helps us answer questions such as "Which students actually improved from exam 1 to exam 2?", "Which students got perfect scores in exam 1?", "How did students who got 45 or better on exam 1 do on exam 2?", and similar questions. These questions are easy to answer for our dataframe since it only has 19 rows, but for larger data sets (such as the `movielens` data), it is not practical to look at each entry separately. As the book explains, the key to answering these type of questions is that R allows indexing by logical vectors.

Let's answer the question "Which students improved from exam 1 to exam 2?" Here we want to make a vector of students whose `improve` variable is positive. We first make a logical vector that is `true` when `improve` is positive and `false` if `improve` is not positive.

```
pos <- exams$improve > 0
pos
```

```
##  [1] FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## [13] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
```

Then we use the logical vector `pos` as the index vector for the ID of students.

```
exams$ID[pos]
```

```
## [1] "3"  "4"  "8"  "17"
```

```
#exams$improve
```

We see that only students 3, 4, 8, and 17 improved from exam 1 to exam 2.

This can be done with just one command, actually.

```
exams$ID[exams$improve > 0]
```

```
## [1] "3"  "4"  "8"  "17"
```

Now subset the entire exams dataset to include only students who improved from exam 1 to exam 2.

```
#Write code here.
exams[exams$improve > 0,]
```

```
##    ID exam1 exam2 improve
## 3   3    39    42       3
## 4   4    46    47       1
## 8   8    41    43       2
## 17 17    47    49       2
```

```
# Using the `stars` data, find out which stars have negative magnitude.
# These are the bright stars.
# Is our sun one of them? --> no the sun has a magnitude of 4.8 according to the data frame

stars$star[stars$magnitude < 0]
```

```
##  [1] "Canopus"          "Arcturus"         "Capella"
##  [4] "Rigel"            "Betelgeuse"       "Achemar"
##  [7] "Hadar"            "Aldebaran"        "Spica"
## [10] "Antares"          "Deneb"            "BetaCrucis"
## [13] "Regulus"          "Acrux"            "Adhara"
## [16] "Shaula"           "Bellatrix"        "Gacrux"
## [19] "BetaCentauri"     "AlNa'ir"          "Miaplacidus"
## [22] "Elnath"           "Alnilam"          "Mirfak"
## [25] "Alnitak"          "Peacock"          "KausAustralis"
## [28] "ThetaScorpii"     "Atria"            "Alkaid"
## [31] "AlphaCrucisB"     "Avior"            "DeltaCanisMajoris"
## [34] "Polaris"          "Mirzam"
```

We can use the same method to figure out how many students (or which students) got a perfect score in exam 1. (You'll have to use `==` instead of `=`, since `==` is a logical operator but `=` is not. Not consistent with `>`, but that's one of R's quirks.)

```
# Write your command here.
exams$ID[exams$exam1 == 50]
```

```
## [1] "6"  "7"  "9"  "12"
```

It is also possible to do the same thing with the `which` command.

```
exams$ID[which(exams$exam1==50)]
```

```
## [1] "6"  "7"  "9"  "12"
```

If you want to find out who got perfect scores on both exams, we can use the logical operator `&` (which is the AND operator).

```
exams$ID[which(exams$exam1==50 & exams$exam2==50)]
```

```
## [1] "6"  "12"
```

We can also find out who got perfect scores on either exam 1 or exam 2, using the logical operator `|` (which is the OR operator).

```
exams$ID[which(exams$exam1==50 | exams$exam2==50)]
```

```
## [1] "6"  "7"  "9"  "12"
```

Suppose we are particularly interested in student #1, student #10, and student #12. We want to see how they did on the exams. We can use the `match()` command.

```
exams$exam1[match(c("1","10","12"),exams$ID)]
```

```
## [1] 48 47 50
```

```
exams$exam2[match(c("1","10","12"),exams$ID)]
```

```
## [1] 48 46 50
```

They did pretty well.

Can you figure out another way to do the same thing, without using the `match()`command?

```
# Write your attempt here.

exams$exam1[exams$ID == "1" | exams$ID == "10" | exams$ID == "12"]
```

```
## [1] 48 47 50
```

```
exams$exam2[exams$ID == "1" | exams$ID == "10" | exams$ID == "12"]
```

```
## [1] 48 46 50
```

Finally, this section also describes the **%in%** command. Note that you use it not via function notation. You use it similar to **+**, in that the first argument goes on the left and the second argument goes on the right.

Here is an example. Suppose you want to find out if anybody got a 42 on exam 1. One way to do this is to ask whether 42 appears in the variable **exam1**.

```
42 %in% exams$exam1
```

```
## [1] TRUE
```

So somebody did get 42 on exam 1. We can ask a similar question for many values simultaneously.

```
c(40:50) %in% exams$exam1
```

```
##  [1] FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
```

So which values appear in exam 1?

We can also use the **subset** command (not described in this section). For example, we can create a new dataframe consisting of students whose score change from exam 1 to exam 2.

```
changed<-subset(exams,exams$improve>0 | exams$improve<0)
changed
```

```
##    ID exam1 exam2 improve
## 2   2    47    46      -1
## 3   3    39    42       3
## 4   4    46    47       1
## 7   7    50    45      -5
## 8   8    41    43       2
## 9   9    50    46      -4
## 10 10    47    46      -1
## 11 11    41    39      -2
## 14 14    47    45      -2
## 15 15    47    31     -16
## 16 16    42    31     -11
## 17 17    47    49       2
## 18 18    46    45      -1
## 19 19    46    39      -7
```

Can you do the same thing using the **not equal** logical operator (which is **!=** and not **!==**)?

```
# Write your code chunk here.
changed <- subset(exams, exams$improve != 0)
changed
```

```
##    ID exam1 exam2 improve
## 2   2    47    46      -1
## 3   3    39    42       3
## 4   4    46    47       1
## 7   7    50    45      -5
## 8   8    41    43       2
## 9   9    50    46      -4
## 10 10    47    46      -1
## 11 11    41    39      -2
## 14 14    47    45      -2
## 15 15    47    31     -16
## 16 16    42    31     -11
## 17 17    47    49       2
## 18 18    46    45      -1
## 19 19    46    39      -7
```

Did any students score a 45 or greater on the first exam AND improve on the second exam? If so, which student(s)?

```
# Write your code chunk here.
exams$ID[exams$exam1 >= 45 & exams$improve > 0]
```

```
## [1] "4"  "17"
```

```
#exams
```

If you are interested in even more practice, work through the exercises in section 2.10 and 2.14 of the textbook:

rafalab.dfci.harvard.edu/dsbook/r-basics.html#exercises-3

rafalab.dfci.harvard.edu/dsbook/r-basics.html#exercises-5